

CSE 259 - Logic in Computer Science

Fall 2024

Recitation-8

Waqar Hassan Khan



Project-2: Task 3

Current code

```
/* ----- */
/* WRITE YOUR CODE FOR TASK-3 HERE */
/* MODIFY THE CODE SO THAT playerA AND playerB AUTO-COMPETE */
/* ----- */

play(Board) :-
    /* move playerA */
    /* get_command asks the user for the move to be made.
       | modify this so that playerA moves on its own */
    . . . get_command(Command),
    . . . execute_command(Command, Board, NewBoard),

    . . . /* move playerB */
    . . . execute_command(playerB, NewBoard, NextNewBoard),
    . . . play(NextNewBoard).
```

Project-2: Task 3

- There are 3 execute_command that are called from play()
- The first one is for inputs taken from user. The second one is for automated players. And the third one is for handling unexpected situations.

```
/* execute the move selected */
execute_command(Move, Board, NewBoard) :-
    parse_move(Move, From, To),
    move(Board, From, To, white, Piece),
    make_move(Board, From, To, NewBoard), !.

execute_command(Player, Board, NewBoard) :-
    respond_to(Player, Board, NewBoard), !.

execute_command(X, Board, _) :- ... % Use to catch unexpected situations
    write('What?'),
    halt(0).
```

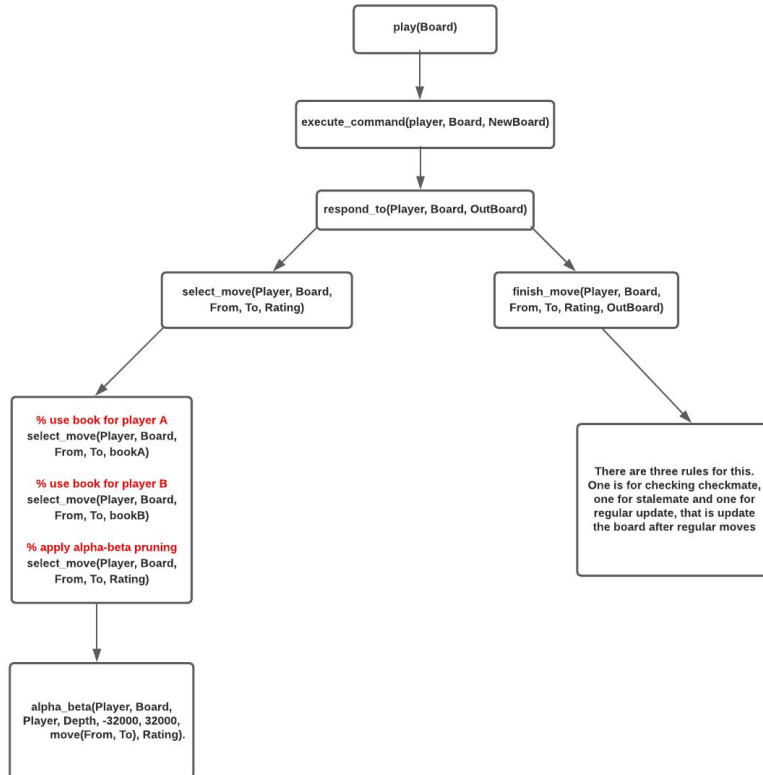
Project-2: Task 3

- Change like the following so that playerA plays on its own

```
/* ----- */
/* WRITE YOUR CODE FOR TASK-3 HERE */
/* MODIFY THE CODE SO THAT playerA AND playerB AUTO-COMPETE */
/* ----- */

play(Board) :-
    /* move playerA */
    /* get_command asks the user for the move to be made.
    | modify this so that playerA moves on its own */
    ... execute_command(playerA, Board, NewBoard),
    ... /* move playerB */
    ... execute_command(playerB, NewBoard, NextNewBoard),
    ... play(NextNewBoard).
```

Project-2: Task 2



- Play called `execute_command`
- `execute_command` calls `respond_to`
- `respond_to` calls 2 rules
 - `select_move`: select a move
 - `finish_move`: finish moving the selected move
- `select_move` has 3 rules. One is for playerA. Start from `select_move` of playerB and start mimicking the code

Project-2: Task 2

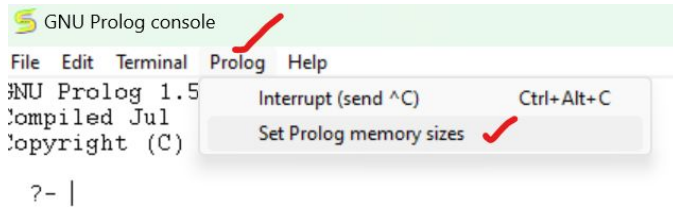
- Look carefully - bookB in select_move appears 3 times. bookA appears 2 times. So, we start working from here

```
finish_move(Player, NewBoard, From, To, Rating, OutBoard) :-
    select_move(Player, Board, From, To, bookB) :-... % Use book for playerB
    player(Player, black),
    bookB(Board, From, To), !.
select_move(Player, Board, From, To, Rating) :-... % time for ALPHA-BETA
    (player(Player, white) -> ply_depthA(Depth);ply_depthB(Depth)),
    alpha_beta(Player, Board, Player, Depth, -32000, 32000,
    move(From, To), Rating).
```

```
-templates > Chess > chess_solution.pl
finish_move(Player, NewBoard, From, To, Rating, OutBoard) :-
    select_move(Player, Board, From, To, bookA) :-... % Use book for playerA
    player(Player, white),
    bookA(Board, From, To), !.
```

Project-2

- Change memory if your program is unexpectedly quitting



Memory sizes setting

You can adjust the size of GNU Prolog memory using environment variables and/or registry keys. Here you can modify registry values. For stacks, sizes are in kilobytes (Kb). For the atom table, it is the number of atoms.

	Default size	Var/Key name	Env. var. value	Registry value	Current size
Trail Stack (undo)	16384	TRAILSZ		284000	284000
Cstr Stack (constraints)	16384	CSTRSZ			16384
Global Stack (heap)	32768	GLOBALSZ		1500000	1499997
Local Stack (control)	16384	LOCALSZ		284000	284000
atom table	32768	MAX_ATOM			32768

New sizes will be taken into account at the next restart.

OK Cancel

Project-2

- For Linux or mac use:

```
export LOCALSZ=284000;export GLOBALSZ=1500000;export  
TRAILSZ=284000;gprolog
```